White Paper

Protecting Users from Firesheep and Other Sidejacking Attacks with SSL

Protecting Users from Firesheep and Other Sidejacking Attacks with SSL

CONTENTS

Introduction

The recent release of the Firesheep Wi-Fi attack tool has increased awareness among both users and attackers of the inherent insecurity of unprotected HTTP connections. Users on unprotected networks who connect to websites through plain HTTP connections expose their connections to those sites to open surveillance and full compromise.

Firesheep allows an attacker connected to the local network to monitor the web sessions of other users on that network. The attacker can then also commandeer the sessions of others, acting in their user context.

Firesheep specifically targets open Wi-Fi networks, but the problem is the same on conventional wired Ethernet networks.

None of this is new. These problems have been generally known, at least in the security community, for years. Firesheep has opened the vulnerability up to others and put devastating identity theft attacks in easy reach of even casual hackers.

As experts proclaimed in reaction to Firesheep, the best solution to the problem is to use TLS/SSL for all connections to websites, including the home page. Perhaps owing to the increased need for processing power it would entail, many large sites have been sparing in their use of TLS/SSL, but such frugality is increasingly indefensible in the face of the level of threats and true costs.

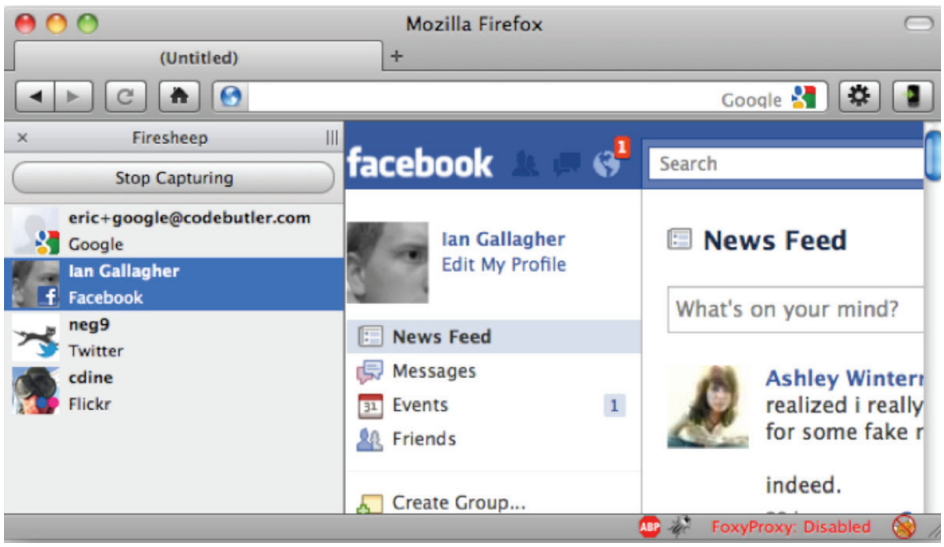
The Problem of Unsecured Wi-Fi

802.11 wireless networking has a troubled history with respect to security. The earliest security mechanism, WEP or Wired Equivalent Privacy, was always difficult to use and eventually turned out to be fatally flawed. Any WEP implementation can be hacked easily with free tools. Mostly because of the ease of use problems, it became very popular to leave Wi-Fi access points wide open with no encryption or authentication at all.

Even though effective and easy-to-use protection, in the form of WPA2 (Wi-Fi Protected Access version 2), is now ubiquitous, open Wi-Fi is still quite common, even for installations which appear to be secure.

For instance, users typically access public Wi-Fi networks by connecting to the local network and then authenticating for full Internet access on a web page served by a local router. The data may be encrypted from the router to the rest of the Internet (although it probably is not), but the local network in and around the coffee shop is still wide open and unencrypted. As a result, any traffic from the clients there to the Internet is unencrypted. The only comprehensive way for users to protect themselves on such a network is to use a virtual private network.

Most of the traffic on public Wi-Fi networks (and many private ones as well) is HTTP, originally the application protocol of the Web, but widely used for many applications. Unless the local and server applications have implemented some sort of private encryption protocol, which is atypical, HTTP is unencrypted.



All traffic is in plain text on the local network and anyone on that same network can read it.

The problem is exacerbated by common practices of websites with cookies. HTTP is stateless and sessionless, which means that individual HTTP commands are independent and unconnected by the server itself. It is up to applications running on the server to keep track of users and their session data, such as the document they are viewing and what they're doing with it.

The standard way to keep track of such things is with cookies, which are data stored locally by the client and associated with a particular server or domain. The server sends these cookies to the client and they are sent back unchanged by the client to the server. The cookies can contain sensitive personal data or other identifying information which can be used by the server to identify the client. Cookies which are used specifically to track user sessions are called session cookies.

Cookies can be sent with a "secure" flag which tells the browser only to send it over an HTTPS (TLS/SSL) session. It is common for websites to encrypt the login process, but not use the secure flag for session cookies. An attacker monitoring an open network can see not only the data sent between the server and client, but also the data in the unsecured cookies. The cookie data then can be used to spoof the user with an attack technique known as sidejacking, which is one form of session hijacking. This is what Firesheep does.

Firesheep is an extension for the Firefox web browser developed by Eric Butler and released in October, 2010 at ToorCon 12, a hacker conference in San Diego. It uses a packet sniffer to intercept unsecured cookies. It displays the names of users on the local network and the services to which they are connected. The attacker can connect to those services using the victim user's credentials by double-clicking on the name.

Self-Protection

An intelligent, resourceful, and motivated user could protect themselves against such threats. After Firesheep was revealed, numerous technical options were presented, none of which are accessible to lay users or provide good protection by default.

One was a Firefox add-in from the Electronic Frontier Foundation called HTTPS Everywhere. This program turns HTTP requests from the browser into HTTPS requests. This works – sort of – with sites which have SSL available but default to HTTP.

As the HTTPS Everywhere page itself explains, it has some problems. For example, it only works with the Firefox browser, it prevents connections to some wireless networks, and users lose access to many Google services. But even if a user is happy with this configuration, it is still possible that the site will use JavaScript to transmit cookies in plain-text HTTP.

Where open Wi-Fi networks are concerned, it would be a lot more secure simply to implement WPA2 with a shared password and put a sign up that says "The password is xxxxxxx' or to make the network name 'coffeehouse pw is xxxxxxx'. This would allow anyone to get on the network, just as with an open network, but WPA2 also implements user isolation so no packet sniffing would be possible. Unfortunately, websites can't count on open network administrators to do this.

A better solution is a virtual private network, which provides an encrypted tunnel from the client system to some other point on the Internet from which all client communications are proxied. While there are consumer VPNs available, they are not well-known, are difficult to use, and can degrade performance, especially on latency-sensitive applications like VOIP and video.

But none of these solutions can possibly be adequate because they require users to take affirmative measures to protect themselves. No matter how much you educate them about it, many users will fail to take these measures and then still be surprised when they are compromised.

Until recently, because of its popularity, Facebook was the most prominent example of the problem of unsecured websites. In late January, 2011 the company announced[1] an option for users to turn on HTTPS for all Facebook communications. They did not make HTTPS the default, though, and few users will turn the option on. It is possible that Facebook is viewing this as a test period and that they will eventually make HTTPS the rule site wide.

The Solution: TLS/SSL Site Wide

Sites which use TLS/SSL site wide are immune to sidejacking. TLS/SSL is, in fact, the only good solution to the problem. Eric Butler himself puts it this way: "The only effective fix for this problem is full end-to-end encryption, known on the web as HTTPS or SSL."

[1] http://blog.facebook.com/blog.php?post=486790652130

Full use of TLS/SSL means full use of the secure flag for cookies. For all practical purposes, all attacks based on packet sniffing fail against a site protected in this way.

Sites that commit to use TLS/SSL for the safety of their users then also need to use a trusted Certificate Authority. Full HTTPS authentication with a trusted CA is the only way for users to know for sure who they are dealing with. Note that Facebook's new SSL feature is a user option and therefore can only be applied after the user has logged in. Since Facebook does not use SSL by default for their home page, users are not completely protected from home page spoofing or eavesdropping.

Costs vs. Benefits of TLS/SSL

Expertise in and software support for TLS/SSL are ubiquitous, leaving cost as the only reason for a site not to use HTTPS for all its connections. What are the costs? There are two, broadly speaking.

First, there is the cost of the certificates themselves. While it is not zero, this cost is at least fixed and predictable. The real wild card to most companies is the added hardware costs. Using TLS/SSL means that all traffic will be encrypted at one end and decrypted at the other. This adds computation which otherwise wouldn't be necessary. On a major site it might seem that this could lead to a substantial hardware cost.

But this may not be the case. Consider the example of Google, which switched their highly popular Gmail service entirely to HTTPS in early 2010. According to Adam Langley, a software engineer at Google, working on OpenSSL, NSS, and Google Chrome, "In January this year (2010), Gmail switched to using HTTPS for everything by default. Previously it had been introduced as an option, but now all of our users use HTTPS to secure their email between their browsers and Google, all the time. In order to do this we had to deploy no additional machines and no special hardware. On our production front-end machines, SSL/TLS accounts for less than 1% of the CPU load, less than 10KB of memory per connection and less than 2% of network overhead. Many people believe that SSL takes a lot of CPU time and we hope the above numbers (public for the first time) will help to dispel that."

This really isn't so surprising, as historical trends in CPU performance have played right into the performance needs of an SSL web server. Not only does Moore's Law continually increase raw performance, but it has resulted in multiple CPU cores on each CPU and large caches.

Multi-core CPUs don't always improve application performance, but for an SSL web server they do because each connection can be split off to a single core. A non-SSL web server, which acts essentially as a simple file server, may not even be taxing the CPU all that greatly. The result is that adding SSL may have little impact on overall system performance. This could explain Google's results. As Langley reiterates: "SSL/TLS is not computationally expensive any more."

He also discusses other measures one can take to optimize the network

performance of a site. Certificate management can have a significant impact. Misconfiguration can lead to performance delays. If, as in the case of Google, a site is willing to optimize its web server or OpenSSL itself, they can improve network performance significantly.

On the other side of the equation is the benefit of securing all communications against certain classes of attacks. This actively protects the site and its users against embarrassing and damaging incidents and sends a message to all that they take security seriously.

Banks figured this out a while ago. A few years ago there were still banks using non-SSL web pages on their home pages and other pages where the user was not yet logged in. The major banks do not do this anymore. They may not have been vulnerable to Firesheep, but they were vulnerable to other forms of attack involving social engineering.

Conclusion

It is important for public websites to provide security configuration features for users to help them manage the accessibility of their personal information, but it is all for naught if someone at the next table can co-opt the account entirely. Security on the Web today begins with a secure connection that is private and cannot be spied on.

Security experts have learned in no uncertain terms that sites cannot expect users to protect themselves. Where systemic solutions for security problems are available to infrastructure and service providers, the bang for the buck is significant. In this way, using TLS/SSL for your entire site will not only harden it against many attacks, but assure your users of that security.

Learn More

For more information about Symantec SSL Certificates, please call
1 (866) 893-6565 Option 3, or 1 (650) 426-5112 or email: isales@symantec.com

More information

Visit our website

http://go.symantec.com/ssl-certificates

To speak with a Product Specialist in the U.S.

Call 1 (866) 893-6565 or 1 (650) 426-5112

To speak with a Product Specialist outside the U.S.

For specific country offices and contact numbers, please
visit our website.

About Symantec

Symantec is a global leader in providing security, storage, and systems management solutions to help consumers and organizations secure and manage their information-driven world. Our software and services protect against more risks at more points, more completely and efficiently, enabling confidence wherever information is used or stored.

Symantec Corporation World Headquarters

350 Ellis Street
Mountain View, CA 94043 USA
1 (866) 893 6565
www.symantec.com

Symantec™

Norton™ SECURED
powered by VeriSign